CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

Fall 2019

Notice

- HW7 posted. Will be due on next Sunday midnight.
- Midterm_Exam1 will be held on:
  - Time: Oct 23rd (Wednesday)
  - Location: the same classroom
  - Topics:
    - The 1st week to this week's topics

Recap

- Merge Sort

Learning Topics

- Discussion on Quiz 2
- More on Quick Sort

Quick Sort

- Basic idea:
  - Partitioning (rearrange) the array into two subarrays, and then sort each subarray independently and recursively.
  - When the two subarrays are in-sort, the entire array is sorted.
  - Different from Merge sort where the array is divided in half, in quick sort, the position of the partition depends on the contents of the array.

Quick Sort

- Property:
  - Also follow the divide-and-conquer paradigm.
  - Often use recursion
  - An in-place algorithm
  - Running time is nlogn on the average for an array of length n.
  - Popular, and more often than other sorting algorithms.
  - It is Quick, quicker than other sorting algorithms in typical applications.

Quicksort Algorithm

Given an array of *n* elements:

- If array only contains one element, return
- Else
  - pick one element to use as *pivot.*
  - Partition elements into two sub-arrays:
    - Elements less than or equal to pivot
    - Elements greater than pivot
  - Recursively Quicksort two sub-arrays
  - Return results

Quick Sort

- The recursive algorithm

```
public void quickSort(int arr[], int low, int high)
   {
      if (low < high)
      {
         //partition
         int index = partition(arr, low, high);

         // Recursively sort each partition
         quicksort (arr, low, index-1);
         quicksort (arr, index+1, high);
      }
   }
```

Quick Sort

- Two key processes:
  - How to pick a pivot?
    - There are many different ways to choose a pivot, and often we choose to use the first element in the array (subarray).
      - Can always choose the first one
      - Can always choose the last one
      - Choose a median one
  - How to do partition?
    - Goal: partition elements into two sub-arrays:
      - Elements less than or equal to pivot
      - Elements greater than pivot
    - How to do it?

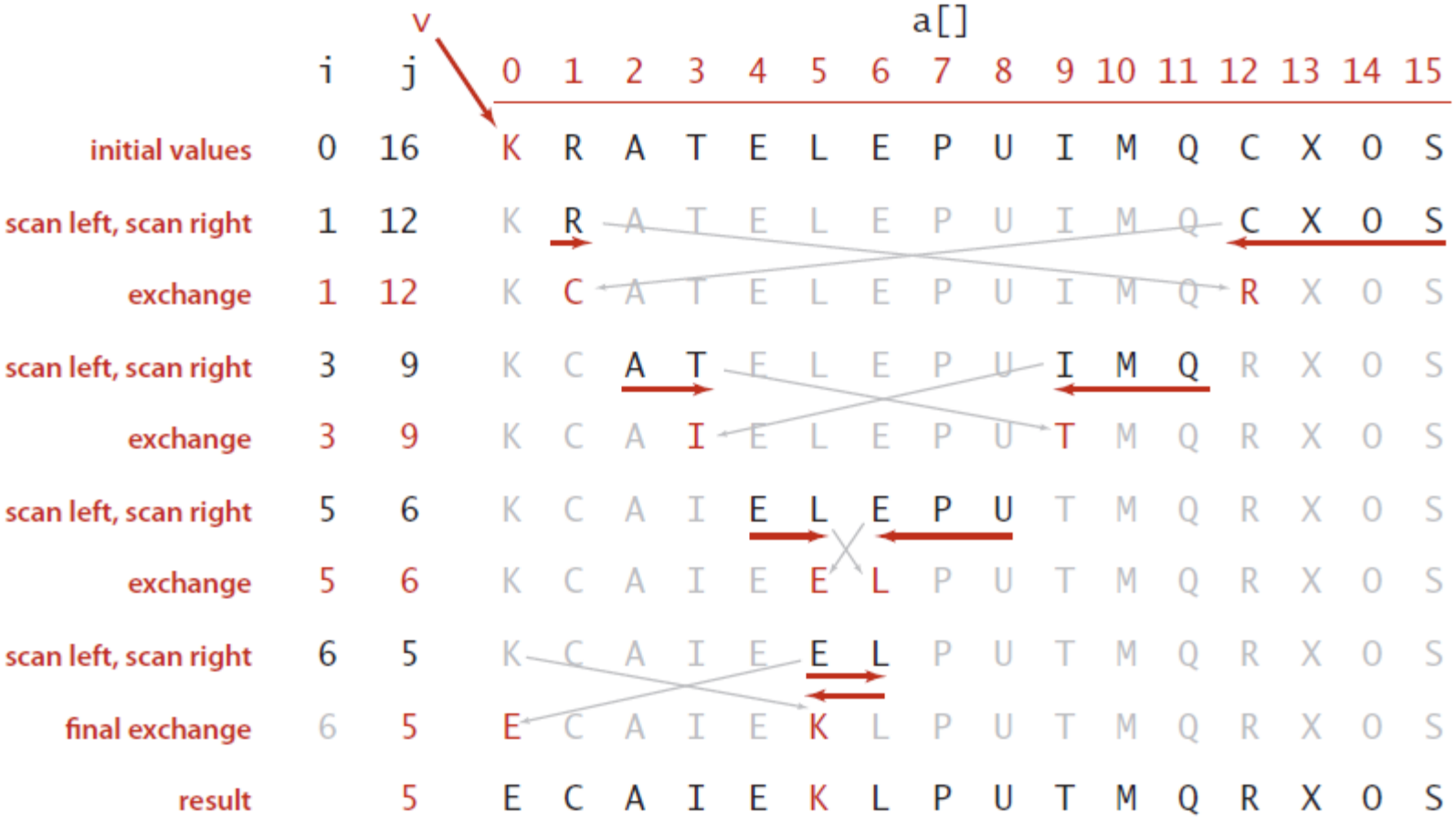
Quick sort: partition code

```
private static int partition(Comparable[] a, int lo, int hi)
{  // Partition into a[lo..i-1], a[i], a[i+1..hi].
   int i = lo, j = hi+1;              // left and right scan indices
   Comparable v = a[lo];              // partitioning item
   while (true)
   {  // Scan right, scan left, check for scan complete, and exchange.
      while (less(a[++i], v)) if (i == hi) break;
      while (less(v, a[--j])) if (j == lo) break;
      if (i >= j) break;
      exch(a, i, j);
   }
   exch(a, lo, j);          // Put v = a[j] into position
   return j;                // with a[lo..j-1] <= a[j] <= a[j+1..hi].
}
```

Example of partitioning

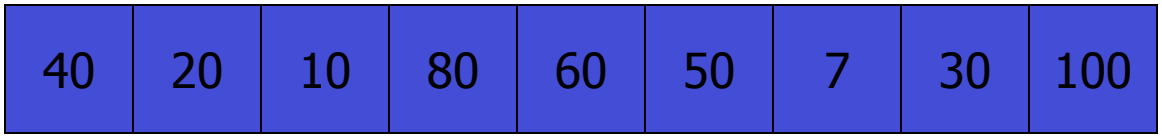
- choose pivot: 4 3 6 9 2 4 3 1 2 1 8 9 3 5 6
- search: 4 3 6 9 2 4 3 1 2 1 8 9 3 5 6
- swap: 4 3 3 9 2 4 3 1 2 1 8 9 6 5 6
- search: 4 3 3 9 2 4 3 1 2 1 8 9 6 5 6
- swap: 4 3 3 1 2 4 3 1 2 9 8 9 6 5 6
- search: 4 3 3 1 2 4 3 1 2 9 8 9 6 5 6
- swap: 4 3 3 1 2 2 3 1 4 9 8 9 6 5 6
- search: 4 3 3 1 2 2 3 1 4 9 8 9 6 5 6 (left > right)
- swap with pivot: 1 3 3 1 2 2 3 4 4 9 8 9 6 5 6

Quick Sort: partitioning

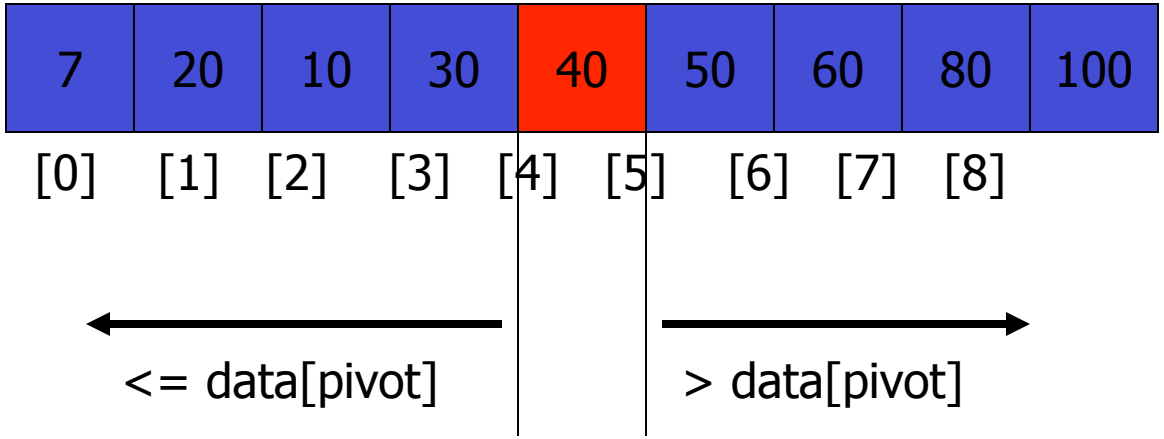


Example

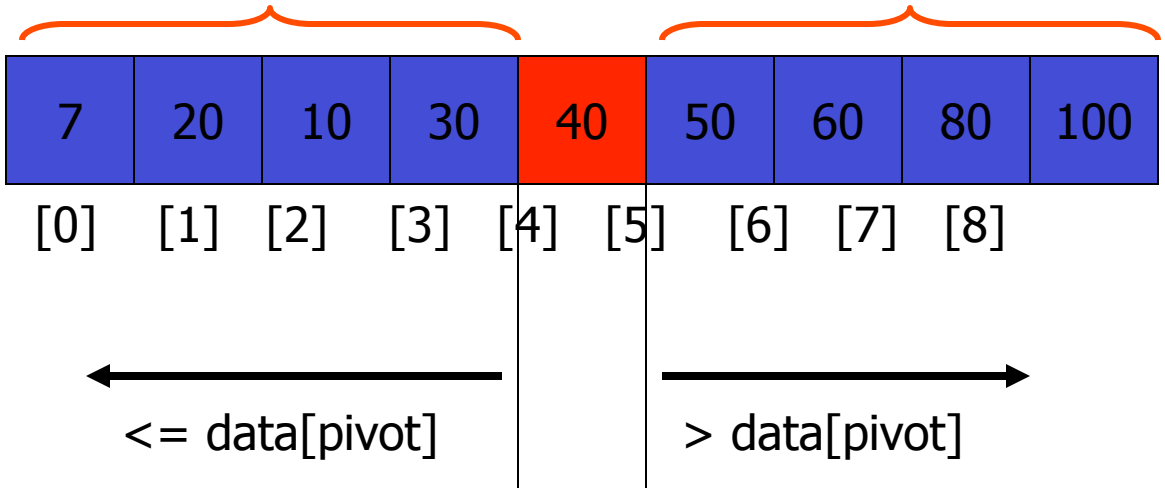
We are given array of n integers to sort:



Partition Result

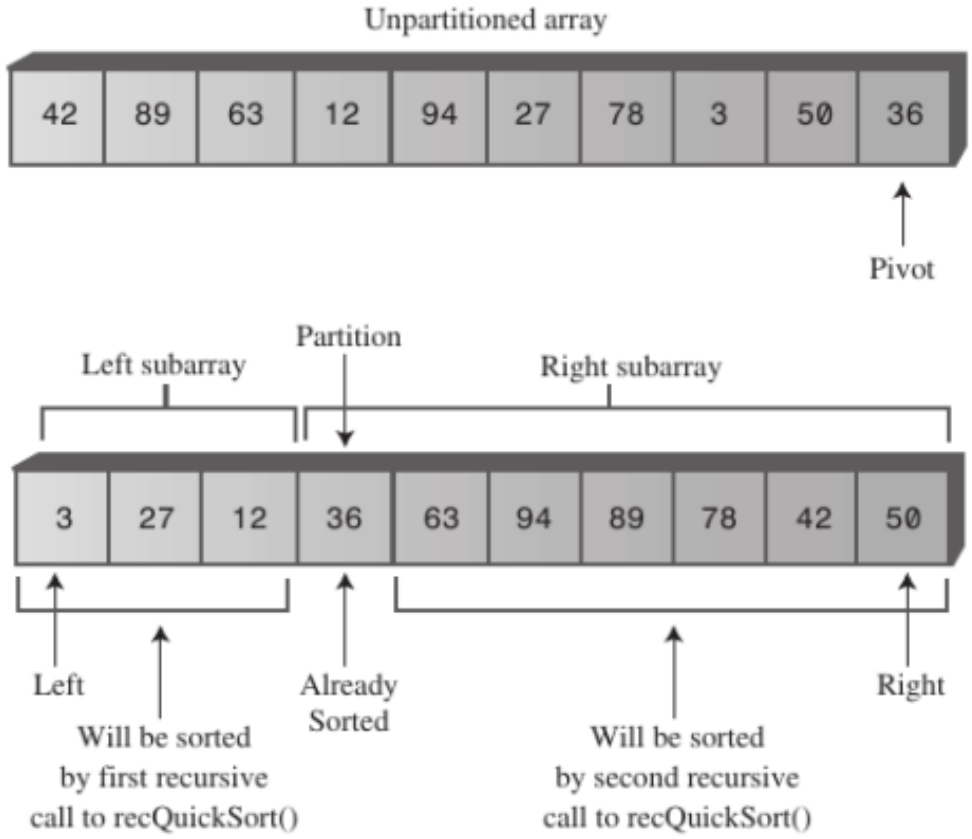


Recursion: Quicksort Sub-arrays



Partitioning the Array (Choosing the last one as pivot)

- Using the last element as the pivot



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 334

Partition(last element as pivot)

```
public int partition(int arr[], int low, int high)
    {
        int pivot = arr[high];
        int i = (low-1); // index determining the end of left partition
        for (int j=low; j<high; j++)
        {
            if (arr[j] <= pivot) // arr[j] should be in the left partition
            {
                i++;
                // swap arr[i] and arr[j]
                int temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }

        // put the pivot in its place
        int temp = arr[i+1];
        arr[i+1] = arr[high];
        arr[high] = temp;

        return i+1;
    }
```

What is the time complexity of partition method?

O(n)

Choosing Pivot

- For simplicity we used the last element as the pivot.
  - fine if the input is random
  - What happens if the array is sorted?
    - We will have a subarray of size 0 and a subarray of size n-1
    - n recursion levels → O(n^2) total complexity
- Ideally we want to divide the array into halves
  - It makes log (n) recursion levels→ O(n log n) total complexity

Quick Sort: Improvement

- Picking a better pivot
  - rather than picking the first/last as pivot:
    - drawback: if the array is already sorted, the partition will be biased to worst case: ~n^2
    - Median of three elements:
  - It will be the best if we can find the median of the entire array, so to evenly partition into two halves
  - Unrealistic, so find the median of three elements for any subarray:
  - The first, middle and the last one
  - Pivot = median of a[0], a[n/2] and a[n-1]
  - a[0]=8, a[n/2]=3, a[n-1]=5, then the median is 5.

Partition(general, pivot isn't the last)

```
public int partition(int[] arr, int left, int right)
{
    int pivot = medianOfThree (arr[], left, right); // Pick pivot point

    while (left<= right) {

      while (arr[left] < pivot)
          left++;

      while (arr[right] > pivot)
         right--;

      // Swap elements, and update left and right
      if (left<= right) {
          int temp= arr[left];
          arr[left]= arr[right];
          arr[right]=temp;
          left++;
          right--;
      }
  }
 return left;
}
```

You can find median of three with comparisons.
It is easier to put the median of the three at the end and use the previous partition code.

Quick Sort: performance analysis

- Best case: partition into evenly half
  - T(N)=T(N/2)+N (n is for the compares in partitioning)
  - proved to be NlogN (Merge sort)

- Worst case: two partitions are extremely biased, i.e. left subarray contains only 1 element, and right one contains the N-1
  - T(N)=N+N-1+N-2+…….1=N(N+1)/2 ~ N^2

Quick Sort: performance analysis

- Average Case: (can be obtained by shuffling)

1. $T_N=N+1+(T_0+T_1+T_2\ldots+T_{N-2}+T_{N-1})/N+(T_{N-1}+T_{N-2}+ \ldots. T_0)/N$
2. $NT_N=N(N+1)+2(T_0+T_1+\ldots..T_{N-2}+T_{N-1})$
3. $(N-1)T_{N-1}=(N-1)N+2(T_0+T_1+ \ldots. T_{N-3}+T_{N-2})$

Subtract 2-3:

4. $NT_N-(N-1)T_{N-1}=2N+2T_{N-1}$
5. $T_N/(N+1)=T_{N-1}/N + 2/(N+1)$
   $=T_{N-2}/(N-1)+2/N+2/(N+1)$
   $=T_{N-3}/(N-2)+2/(N-1)+2/N+2/(N+1)$
   $=\ldots\ldots\ldots$
   $=T_1/2+2(1/3+1/4+1/5+ \ldots.. 1/(N+1)$
6. $T_N \sim 2(N+1)(1/3+1/4+1/5+ \ldots. 1/(N+1)) \sim 2N\log N$

the integral of function 1/x is logx

Stable Sort

- Stability: A sorting method is stable if it preserves the relative order of equal keys in the array.
- Is merge sort stable?
  - yes, items with equal keys appear in the same order as before
- It does not matter when the key=value, like list of integers,…
- Example
  - key=student names, value= section in this course

Summary on Sorting Algorithms

Algorithm	Best Case	Average Case	Worst Case	In-Place?	Stable?
Selection	$O(N^2)$	$O(N^2)$	$O(N^2)$	Yes	No
Insertion	O(N)	$O(N^2)$	$O(N^2)$	Yes	Yes
Bubble	$O(N^2)$	$O(N^2)$	$O(N^2)$	Yes	Yes
Merge	O(NlogN)	O(NlogN)	O(NlogN)	No	Yes
Quick	O(NlogN)	O(NlogN)	$O(N^2)$	Yes	No

Closing Remarks on Sorting

- Hybrid Approach
  - Do quick sort, but switch to insertion sort if array is small (around 10 elements).

- All the algorithms we discussed so far were based on comparisons.
  - You cannot do better than O(n logn) in a comparison based sorting algorithm.

Closing Remarks on Sorting

- There are non-comparison based algorithms
  - Counting sort
    - Gives O(n+k) if elements are in range 1 to k
    - Needs additional space to save counts
  - Radix sort
    - Sort digit by digit from least significant to the most significant digit.
    - Uses counting sort every time.
    - Does it matter to use a stable sort for every digit?
      - Yes, otherwise it may undo the previous sort step. Example: 5, 29, 21

System sort

- import java.util.Arrays;
- int[] a = new int(10);
- Arrays.sort(a);
- Property:
  - Different sorting algorithm for different data types; Typically, 3-way quick sort for primitive data types and Merge-sort for reference type.
  - Trade speed and memory usage for stability (primitive data type) and guaranteed performance (reference type)
  - A method that data types that implements Comparable interface
  - A method that uses Comparator.